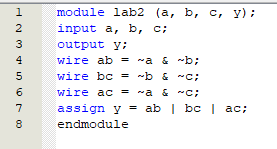
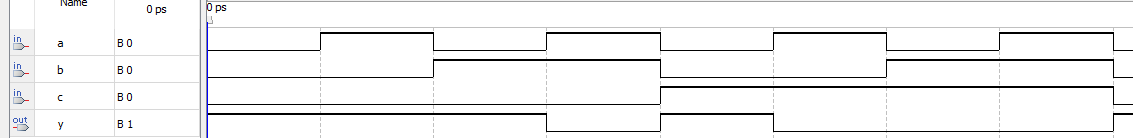
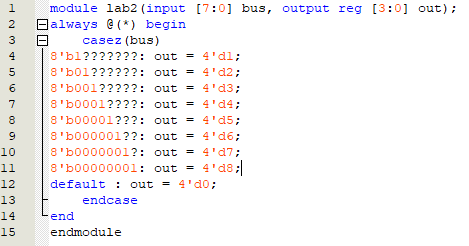
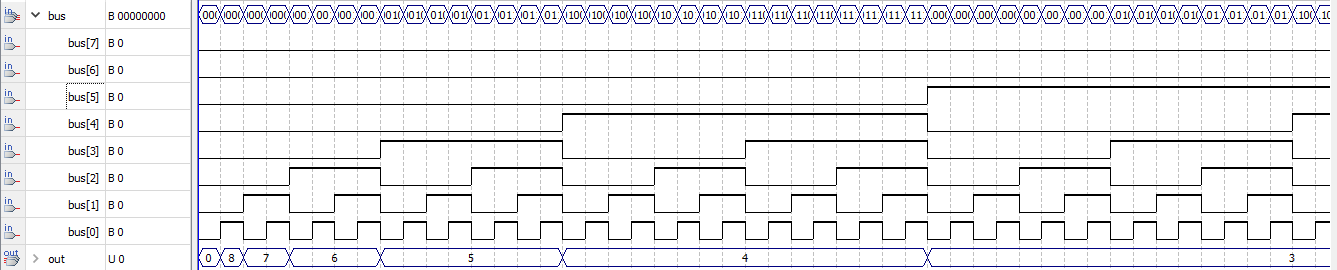
1.

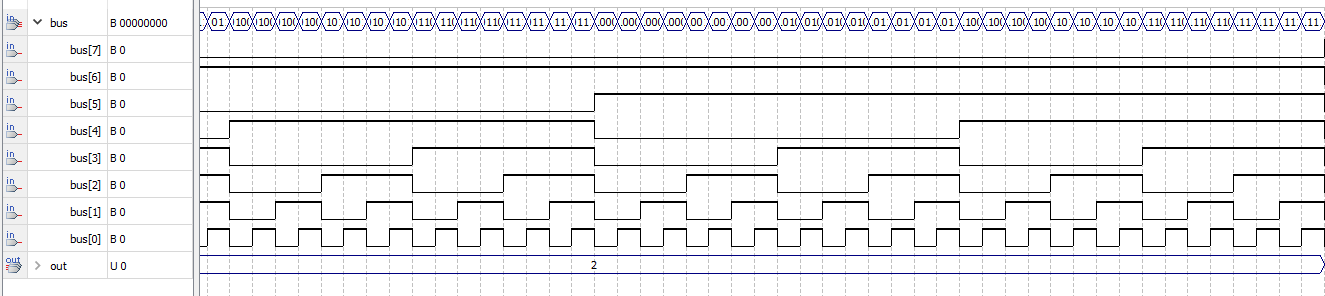


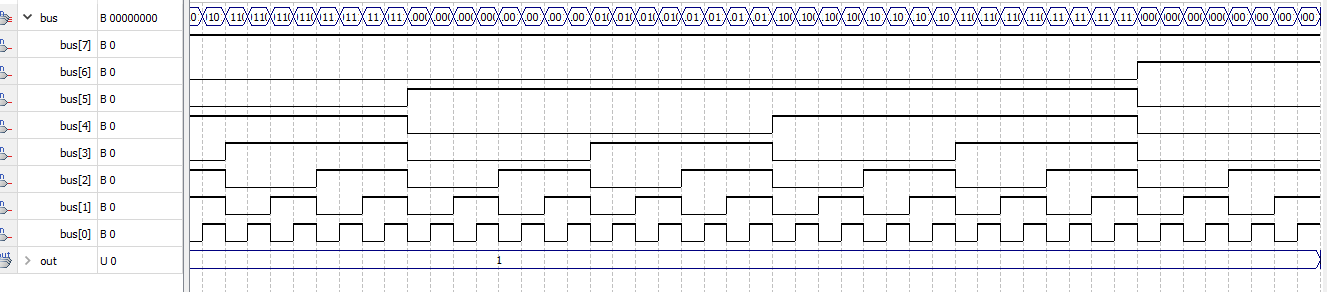


2.









3.

module lab2(stream, clk, parity);

input stream;

input clk;

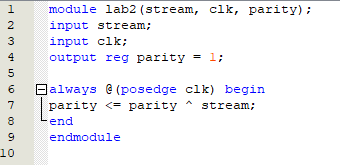
output reg parity = 1;

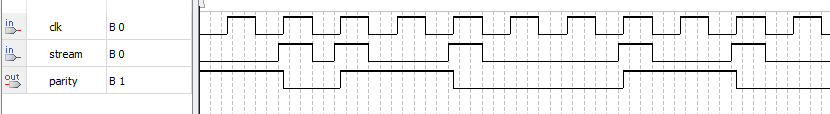
always @(posedge clk) begin

parity <= parity ^ stream;

end

endmodule





4.

module lab2(stream, clk, sequence);

input stream;

input clk;

output reg sequence;

reg [2:0] stack;

always @(posedge clk) begin

if (stream)

begin

stack = stack << 1;

stack[0] = stream;

end

else

begin

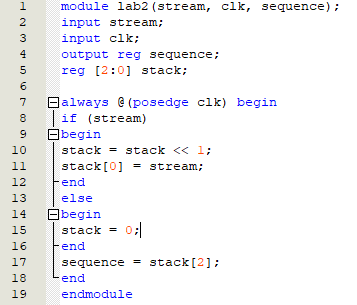
stack = 0;

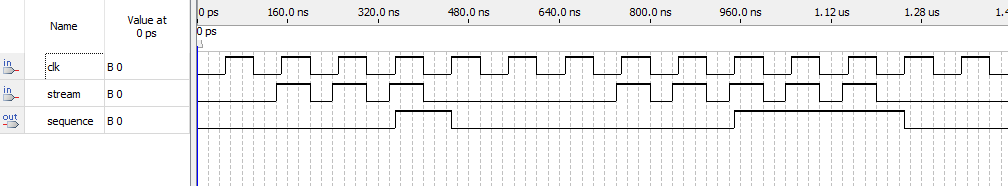
end

sequence = stack[2];

end

endmodule





5.

module lab2(input stream, input clk, input reset, output out);

reg [2:0] state, nextstate;

//encode the state

parameter S0 = 3'b000;

parameter S1 = 3'b001;

parameter S2 = 3'b010;

parameter S3 = 3'b011;

parameter S4 = 3'b100;

parameter S5 = 3'b101;

//state register

always @ (posedge clk, posedge reset)

if (reset) state = S0;

else state = nextstate;

//next state logic

always @ (\*)

case (state)

S0 : if (stream) nextstate = S0;

else nextstate = S1;

S1 : if (stream) nextstate = S2;

else nextstate = S1;

S2 : if (stream) nextstate = S0;

else nextstate = S3;

S3 : if (stream) nextstate = S4;

else nextstate = S1;

S4 : if (stream) nextstate = S5;

else nextstate = S3;

S5 : if (stream) nextstate = S0;

else nextstate = S1;

default : nextstate = S0;

endcase

assign out = (state == S5);

endmodule

